The Performance of Q-Learning within SDN Controlled Static and Dynamic Mesh Networks

Douglas Harewood-Gill*, Trevor Martin†, Reza Nejabati*
*High Performance Networks Group, University of Bristol, Bristol, United Kingdom,
Emails: {douglas.harewood-gill, reza.nejabati}@bristol.ac.uk
†Engineering Mathematics, University of Bristol, Bristol, United Kingdom,
Emails: {trevor.martin}@bristol.ac.uk

***Abstract*—Current infrastructures are reaching the point where existing networking methods are unable to cope with the exponential growth of traffic and Quality of Service (QoS) requirements. New techniques are necessary to keep pace. One such technique, Software-Defined Networking (SDN) uses a central controller to program many individual network devices. However, SDN uses heuristic algorithms that do not always select the optimal path.**

This paper looked at creating three Q-Routing algorithms leveraging SDN and Mesh network topologies. Two algorithms used one network metric each (Latency and Bandwidth) and the third used multiple metrics. Results showed that the single metric Q-Routing algorithms on average performed as well as the K-Shortest Path versions while Q-Routing with multiple network metrics failed to match K-Shortest Path (different combination of metrics means these algorithms are not comparable). Results also showed that Q-Routing was able to calculate paths faster than K-Shortest Path in both static and dynamic networks.

***Index Terms*—SDN, Mesh Network, Reinforcement Learning, Q-Routing, K-Shortest Path.**

I. Introduction

In light of massive growth in IP traffic [1], there is a need for new approaches in network management, including path-calculation. Traditional path calculation uses standard heuristic algorithms such as Bellman-Ford or Dijkstra to find the shortest path, but does not always calculate the optimal path. Q-Learning [2] (QL) uses unsupervised Reinforcement Learning to determine optimal behaviour to maximise performance when interacting with its environment. QL has several advantages; QL does not require a model of the environment, only environmental information to start learning. QL can learn without user feedback [3]. QL can be used for many applications including signal localisation [4], power system controllers [5], network path calculation [6], etc.

The last application, Q-Routing (QR) [6] has been implemented in legacy networks ([7], [8], [9]). However, there are few examples in current literature of QR in Software-Defined Networks (SDN). At the time of writing, QR only used link latency to calculate paths.

Using SDN, this research aims to. 1. Investigate the use of Q-Routing using latency (QRL). 2. Create a version of Q-Routing using bandwidth (QRB). 3. Create a QR algorithm using multiple metrics based on [10].

978-1-7281-5684-2/20/$31.00

II. Background and Related Works

QL [2] works by training Q, a structure of State (S) / Action (A) pairs for a fixed destination or objective. S refers to the current node and A, to any of the neighbour nodes that can be selected. A reward formula is used to train Q; Q_N(S,A) is the new value for a specific S and A pair, Q(S,A) is the value currently held in Q for that pair, α is the learning rate, R(S,A) holds the reward (environmental values) for a specific S and A pair, γ is the discount rate and Q(S',A') denotes all possible actions from the next state.

$$Q_N(S,A) = Q(S,A) + \alpha[R(S,A) + \gamma MaxQ(S',A') - Q(S,A)]$$

To train Q, a random start node is selected and the algorithm finds the A with the best value or if Q is empty, selects an A at random. After each move, Q updates the current Q(S,A) using the reward formula and the selected A becomes the next S. This continues until the destination has been found, completing one training cycle. After multiple training cycles, Q can be used to find a path to the destination.

Chavula et al [10] proposes a method to improve bandwidth usage and reduce latency using QL for path selection in SDN. This combines link latency, link bandwidth and available link bandwidth, tested on a partial mesh network. Results show an increase in throughput when multipath was used and when latency was the primary metric, lower latencies were achieved. Using both latency and bandwidth resulted in throughput, latency and jitter approximately the same as standard approaches.

Kim et al [11] discusses a way to solve congestion caused by Dijkstra's shortest path as it does not consider bandwidth. In an SDN network, QL is trained to select a new path if the congestion threshold on the original path has been reached. A simulated partial mesh network was used. Results show slightly faster transmission times when compared to Dijkstra's shortest path / extended shortest path.

Adaptive Q-Routing Full Echo (AQRFE) [9], based on QR adds a dynamic learning rate on each node in addition to the standard learning rate. The aim is to improve QR exploration where the standard learning rate is used if the Q-value comes from a neighbour node, otherwise the new learning rate is employed. AQRFE was tested on a partial mesh network and compared against QR and Dual QR. In a low load network,

AQRFE achieved a lower average delivery and in a large load network, it was on par with both QR and Dual QR.

Dual Reinforcement Q-Routing (DRQR) [12] was designed to calculate paths to sustain high loads with a low average delivery time. In QR, when Node A requests from all neighbour nodes their latency estimate to the destination, each neighbour sends this information back. Node A then uses information from the estimated shortest path in the reward formula to update its Q table. [12] calls this "Forward Exploration". DRQR adds "Backwards Exploration" which allows Node A to share information about the traversed path with neighbour nodes so they can update their own Q tables. DRQR was tested on a partial mesh network against Shortest Path and QR. Results showed at low load, Shortest Path had smallest average delivery time. DRQR, at low loads performed better than QR and at medium and high loads, better than Shortest Path and QR.

This literature represents the most relevant QR research. We found few examples of QR in SDN with only one using multiple network metrics and none using link bandwidth instead of link latency. With this in mind, this research (i) looks at implementing QR in SDN controlled networks (static and dynamic) using full and partial mesh network topologies of varying sizes and connectivity. (ii) Creates a new derivative of QR using link bandwidth. (iii) Implements a Multi-Variable QR Algorithm (MVQRA) based on[10]. Each QR algorithm is compared against an equivalent K-Shortest Path (KSP) algorithm.

III. Implementing Q-Routing within an SDN Controlled Mesh Network

A. SDN Controller

QL has been widely applied to routing problems in traditional networks but despite the advantages, their use in SDN is limited [13]. The Ryu SDN controller [14] is used to implement each algorithm and can be used in conjunction with OpenFlow emulators such as Mininet and actual OpenFlow compatible forwarding nodes.

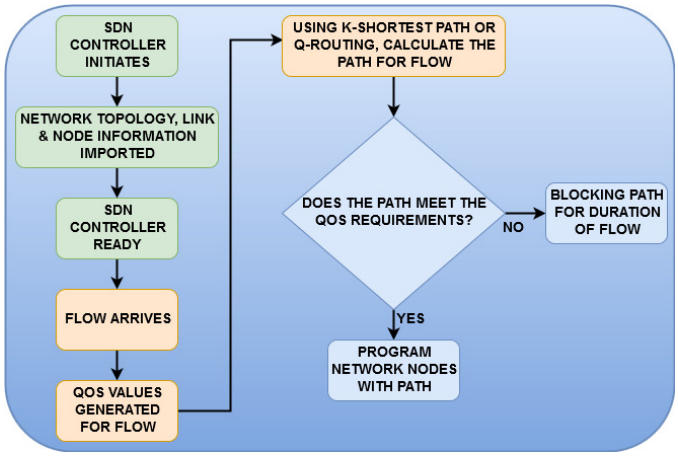

Fig. 1. Flow diagram showing the operation of the SDN controller program.

Ryu works as shown in Fig. 1. The controller maps the network topology and makes ready to receive flows. When a path calculation request is received from a host sending traffic via a network node, random QoS metric/s are generated for that flow to simulate QoS requirements. The path is calculated using QR / KSP and then checked to see if it meets QoS requirements. If it does then the nodes in the path are programmed by the SDN controller, otherwise, the flow is blocked from transmitting for the duration of the flow.

B. K-Shortest Path and Q-Routing Algorithms

Shortest Path is commonly used in both legacy and SDN networks and so is a good baseline to compare the performance of QR. Each QR algorithm has an equivalent KSP algorithm and all algorithms are implemented in the SDN framework. For QRL and QRB, 150 training cycles are required to sufficiently train Q. For MVQRA, 500 training cycles are required. The following algorithms were considered.

- K-Shortest Path using Latency (KSPL):- Based on Yen's K-Shortest Loopless Paths [15]. It finds all paths between the source and destination nodes and returns a list of these paths ranked by their end to end latency, smallest first.
- K-Shortest Path using Bandwidth (KSPB):- Is a variation of KSPL and returns a list of paths ranked by their end to end bandwidth, largest first.
- K-Shortest Path using Latency and Bandwidth (KSPLB):- Is an adaptation of the KSP but uses latency, maximum bandwidth and available bandwidth on a link.
- Q-Routing using Latency (QRL):- Is a variant of QR designed for path calculation [6].
- Q-Routing using Bandwidth (QRB):- Based on QRL, the main difference is in the latter portion of the QL equation. The aim is to find the path with the largest bandwidth by finding the link with the smallest bandwidth on each path (i.e. the bandwidth for the complete path). The QL equation was modified. The maximum is taken over all of N, the neighbour nodes to S.

$$Q_{New}(S,D) = (1-\alpha)Q(S,D) + \alpha \max_{N} (Min(B(S,N), Q(N,D)))$$

At the time of writing, no instances could be found in current literature of QRB.
- MVQRA:- This algorithm included link bandwidth and free link bandwidth capacity in the reward formula, based on [10]. An aggregation function combines link latency (L_l), maximum link bandwidth (B_m) and free link bandwidth (B_f):

$$K = \beta_1 \times \frac{B_f}{B_m} + \beta_2 \times \frac{L_1}{n}$$

where we follow [10] in arbitrarily setting $n = 1000$, and the weights β_1 and β_2 are set to 1. K replaces R, the link latency in the original QR formula resulting in:-

$$Q_{new} = Q(S,A) + \alpha(K + \alpha \min_{A'} (Q(S',A') - Q(S,A)))$$

C. Topology Creation and Traffic Generation

Mininet is used for two functions. First, build and emulate a network from a specified topology using open source code [16]. Second, generate random UDP traffic flows at random intervals using an iPerf server and client on each host node.

IV. Experimental Setup and Evaluation

A. Experiment 1:- Static Path Blocking Performance Tests

Compares the percentage of flows blocked for each QR algorithm and its KSP equivalent. Each algorithm is tested on three topologies, (10 Host / 10 Switch, 25 Host / 25 Switch and 50 Host / 50 Switch). Each topology has variants based on average mesh connectivity (AMC), ranging from a few connections per node to a full mesh. To complete one run, each algorithm calculates the paths for approximately 2,000 flows and the "active" or "blocked" status of this path is recorded. Each run is performed five times on each topology size for each AMC. The blocking % for each run B_p is:-

$$B_p = 100 \times \frac{B}{N}$$

where B is the number of blocked flows and N the total number of flows in the run.

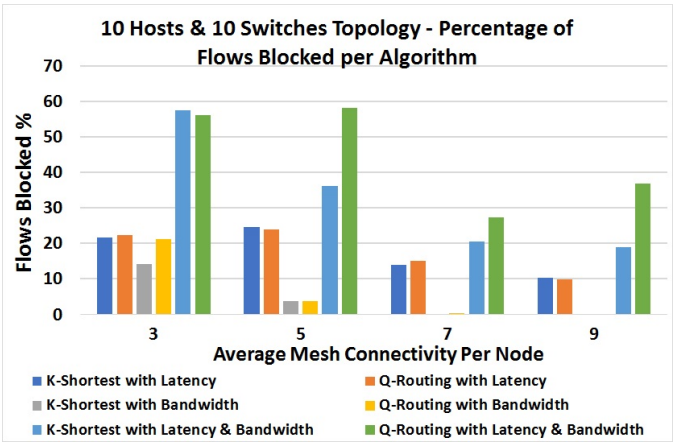

Fig. 2. Flow Blocking Comparison for each KSP and QR Algorithm on 10 Host/10 Node topology.

The graphs in Figs. 2 to 4 show the average percentage of flows blocked for each QR and equivalent KSP algorithms.

B. Experiment 2:- Static K-Shortest Path N-Value Evaluation

Experiment two looked at the effect of altering the number of paths (N) that each KSP algorithm found between the source and the destination nodes. The time taken to calculate a path and the percentage of flows blocked were compared to the QR equivalents. Each test used the same static 50 Host / 50 Node topology with an AMC of 49 with the same traffic generator. Tests were performed on an IBM x3455 server node running 2x Dual-Core AMD Opteron 2218 CPU's, 8GB of Ram and an 80GB Hard drive. An average was taken from 1,000 samples for each value of N for each algorithm.

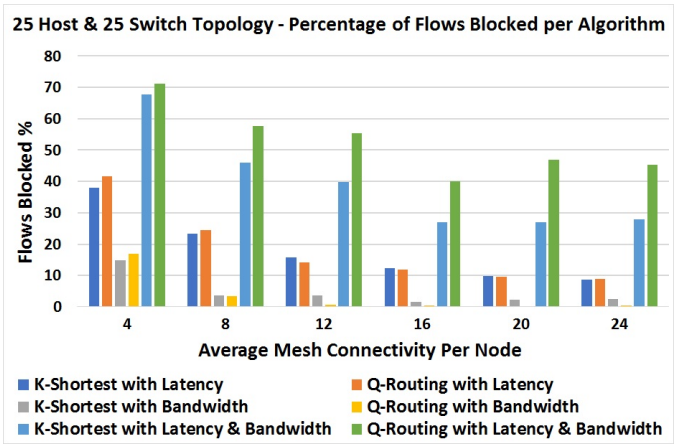

Fig. 3. Flow Blocking Comparison for each KSP and QR Algorithm on 25 Host/25 Node topology.

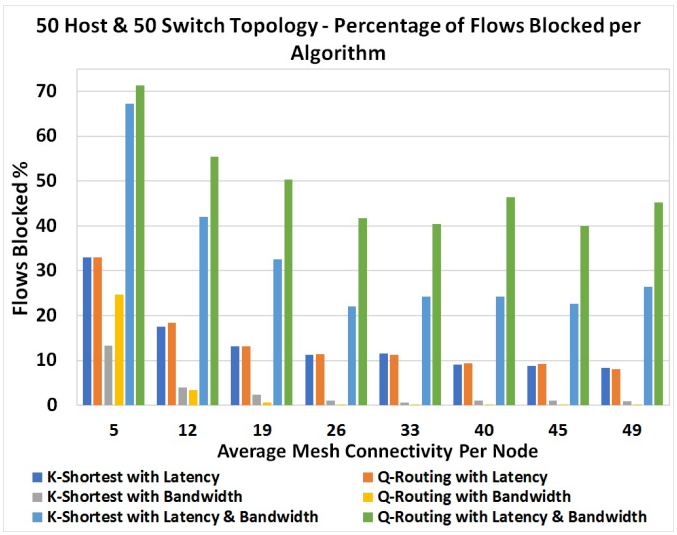

Fig. 4. Flow Blocking Comparison for each KSP and QR Algorithm on 50 Host/50 Node topology.

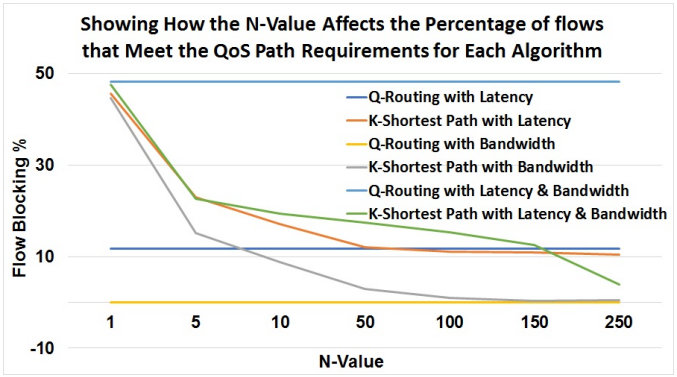

Fig. 5. Relationship between N and the blocking performance

Figs. 5 to 6 show that as N increased, the percentage of flows being blocked decreased and the time taken to calculate a path between increased.

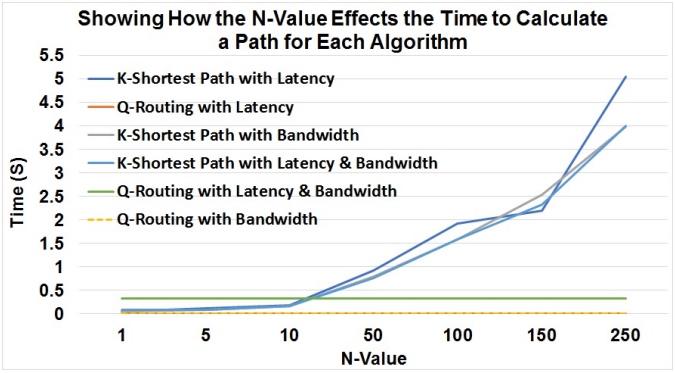

Fig. 6. Relationship between N and the time taken to calculate a path

C. Experiment 3:- Dynamic, Adaptive Q-Routing Performance

Initially Q was trained for each destination node in the network as soon as the SDN program started. However, for a dynamic network, an additional training cycle was added for every incoming flow request allowing Q to update and adapt to network changes and calculate a path between source and destination hosts using Q that reflected the changing topology. Two tests were performed to see how effective QR was at adapting compared to KSP.

Test 1: A small partial mesh Mininet topology was created with one source and destination host, H1 and H2 respectively (Fig. 7). The QoS value for latency and bandwidth were fixed for each incoming flow to 45ms and 80Mb. There were only two paths that could be utilised for each flow to meet the QoS requirements, (S1-S2-S3-S4-S5) and (S1-S6-S10-S11-S6). The first path being optimal. Each flow was created on H1 and sent to H2. To see how well QRL and QRB could adapt, links were broken and restored at intervals forcing QR to adapt and find an alternative path. The first break was set to a flow count of 10 with each subsequent link restore or link break taking effect in 50 flow increments thereafter.

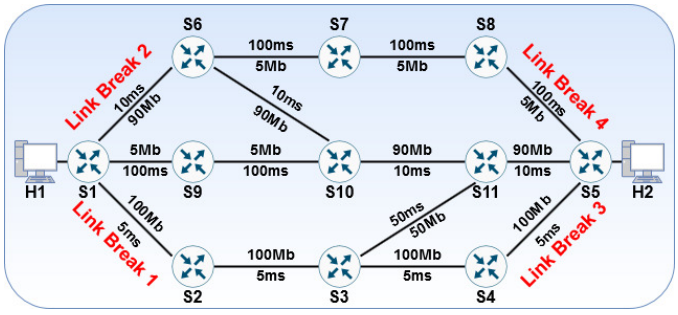

Fig. 7. A topology to test QRs ability to adapt to dynamic networks.

The results (Fig. 8) shows that QR adapted very quickly to a changing topology.

Test 2:- Using the same topology and traffic generator as Experiment 2 and a fixed value of N = 250 for KSP, this test measured how long adaptive QR took to calculate a path compared to KSP. A mechanism was introduced to break and restore links at random in the topology with up to four broken at any one time.

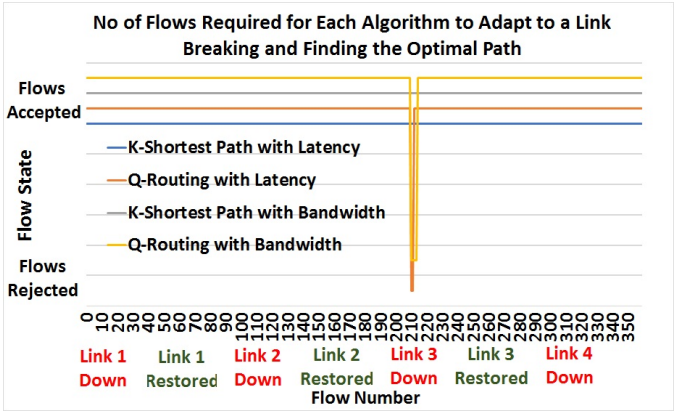

Fig. 8. QR Adapting to link breaks and restores compared to KSP on (Fig. 7).

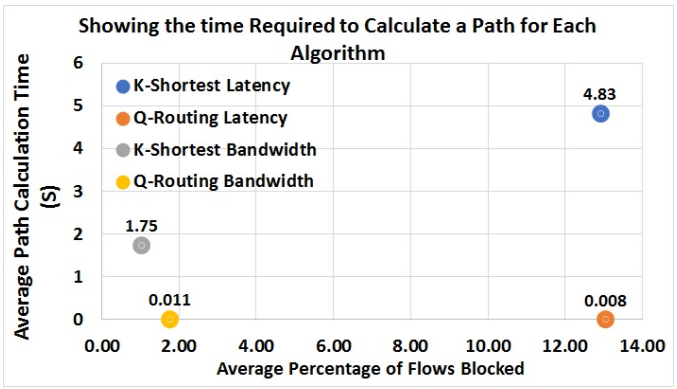

Fig. 9. Path calculation time VS percentage of flows blocked.

Fig. 9 shows the percentage of flows blocked by QR differed little from KSP equivalents. The time difference for path calculation was far more pronounced with QRL taking on average 0.008s compared to KSPs 4.83s and QRBs 0.011s to KSPs 1.75s.

D. Evaluation

To be considered as a viable alternative or supplement to KSP, the performance of QR needed to be comparable. For experiment one, both QRL and QRB had performance virtually identical to their KSP counterparts, although, KSPL on average had a marginally lower blocking percentage than QRL.

QRB blocked a larger percentage of flows on each of the three different topologies when AMC was at its lowest. However, as soon as the AMC increased, performance mirrored that of KSPB. This suggests that QRB would be better utilised in topologies with a medium to high AMC value.

MVQRA, based on [10] was able to calculate paths but blocked a higher percentage of flows than the KSP equivalent. There were two reasons for this; first, the aggregate function took the values of multiple metrics and generated one value that was used in the reward formula to calculate Q. After

aggregation, there was no way in the pathfinding phase using Q to establish if the aggregated value meant a link with low latency and high bandwidth or vice versa. In contrast, the KSP equivalent algorithm treated latency and bandwidth separately so the results are not entirely comparable. Second, the reward formula used "min" to find the lowest value of Q over all neighbour nodes. This was not compatible with bandwidth.

Experiment two results showed that an N-Value between 150 to 250 was required for each KSP algorithm to match or surpass the performance of the QR equivalent but resulted in noticeably longer times to calculate paths. i.e. when N=250, KSPB took on average 3.99s to calculate a path compared to QRB which took 0.012s. Reducing N reduced the path calculation time for KSP. For N=5, KSPL took 0.11s to calculate a path, faster than N=250 but still slower than QRLs 0.008s. However, the percentage of flows blocked increased from 10.34% to 23.04%, worse than QRLs 11.64%. It should be noted that these tests were performed on a static topology where it is possible for KSP to pre-process all paths instead of calculating them in real time [17].

Experiment three introduced a dynamic element to the topology (random link failures) so QR had to adapt to meet QoS requirements for each new flow and could act as a more realistic comparison to KSP unable to use pre-processing [17]. QR, by contrast, could initially train for each destination node and then re-train for a single cycle for each incoming flow. The results show that QRL and QRB were quick to adapt on the small topology (Fig. 8) though both algorithms did block flows while adapting to topology changes. By comparison, the difference between the percentage of flows blocked in the larger topology between KSP and QR was minimal while the difference in average path calculation time was substantial. QRL took 0.008s on average to calculate a path compared to KSPLs 4.83s and QRB took 0.011s compared to KSPBs 1.75s.

Both QRL and QRB had similar performance to the KSP equivalent algorithms in terms of blocking percentages and calculated paths faster than KSP in both static and dynamic network topologies. For scalability, the time for KSP to calculate a path increased exponentially as the topology size and the AMC increased. Using a smaller N-Value allowed for faster path calculation times but reduced performance. In contrast, not only was QR faster, Hoceini [7] and the results show that it is scalable. This could allow QR to become a viable replacement for KSP in larger network topologies or topologies with a high AMC. Finally, [12] showed that QR and other variants calculated the optimal path faster than Shortest Path in legacy networks with high traffic loads. These results are consistent with the results shown here.

V. Conclusion

This paper proposed three objectives. 1. Implementation of an existing form of QRL, 2. Creation of a new algorithm, QRB, 3. Implementation of a QR adaptation (MVQRA) that aggregated max link bandwidth and free link bandwidth capacity in addition to latency. All algorithms were designed for path calculation within an SDN network.

Results show that QR using single network metrics gives near equal performance for path calculation to KSP. QR is much faster at path calculation in both static and dynamic networks. While MVQRA did not perform as well as its KSP equivalent, the results show QR is scalable, flexible and more efficient. QR should be considered a serious area of research for the next generation of path calculation algorithms either to enhance or replace existing heuristic algorithms.

Acknowledgements

We thank the Engineering and Physical Sciences Research Council Centre for Doctoral Training in Communications (EP/I028153/1 and EP/L016656/1) and Roke Manor for financial assistance and support.

References

[1] Cisco. (2018) Vni complete forecast highlights. [Online]. Available: https://www.cisco.com/c/m/en_us/solutions/service-provider/vni-forecast-highlights.html

[2] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[3] R. While. (2019) Artificial intelligence: Reinforcement learning. [Online]. Available: http://teaching.csse.uwa.edu.au/courses/CITS4211/Lectures/wk7.pdf

[4] S. Wu, "Illegal radio station localization with uav-based q-learning," *China Communications*, vol. 15, no. 12, pp. 122–131, Dec. 2018.

[5] P. K. N. Zhou, A. Swain and N. C. Nair, "A dynamic fuzzy q-learning controller to improve power system transient stability," *IEEE International Conference on Power System Technology (POWERCON)*, pp. 1–6, Sep. 2016.

[6] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Proceedings of the 6th International Conference on Neural Information Processing Systems*, pp. 671–678, 1993.

[7] S. Hoceini, A. Mellouk, and B. Smail, "Average-bandwidth delay q-routing adaptive algorithm," *IEEE International Conference on Communications*, pp. 1840–1844, May 2008.

[8] A. Kavalerov, Y. Shilova, and Y. Likhacheva, "Adaptive q-routing with random echo and route memory," *20th Conference of Open Innovations Association (FRUCT)*, pp. 138–145, Apr 2017.

[9] Y. Shilova, M. Kavalerov, and I. Bezukladnikov, "Full echo q-routing with adaptive learning rates: A reinforcement learning approach to network routing," *IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, pp. 341–344, Feb. 2016.

[10] J. Chavula, M. Densmore, and H. Suleman, "Using sdn and reinforcement learning for traffic engineering in ubuntunet alliance," *International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pp. 349–355, 2016.

[11] S. Kim, A. Son, and C. Hong, "Congestion prevention mechanism based on q-leaning for efficient routing in sdn," *International Conference on Information Networking (ICOIN)*, pp. 124–128, 2016.

[12] S. Kumar and R. Miikkulainen, "Dual reinforcement q-routing: An on-line adaptive routing algorithm," *Smart Engineering Systems: Neural Networks, Fuzzy Logic, Data Mining, and Evolutionary Programming*, vol. 7, 1997. [Online]. Available: http://nn.cs.utexas.edu/?kumar:annie97

[13] D. Kreutz, F. Ramos, P. Veríssimo, and C. Rothenberg, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[14] Ryu. (2019) Component-based software defined networking framework. [Online]. Available: https://osrg.github.io/ryu/

[15] Y. Yen, "Finding the k shortest loopless paths in a network," *Manag. Sci*, vol. 17, no. 11, pp. 712–716, Oct. 1971.

[16] G. Bernstein. (2019) Grotto networking:- sdn fun. [Online]. Available: https://www.grotto-networking.com/SDNfun.html

[17] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 15–26, Aug 2013.